Database - Bug #7388

Create server configuration container for cache sizes

05/29/2023 03:17 AM - Alexandru Lungu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Dănuț Filimon	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Database - Bug #7330: Increase psCache size	**Closed**
Related to Runtime Infrastructure - Bug #7425: Avoid directory usage in stati...	**Test**
Related to Database - Feature #6815: configure all cache sizes in the directo...	**Review**

History

#1 - 05/29/2023 03:25 AM - Alexandru Lungu

Make FWD caches configurable in size. As a place to start, consider only persistent related caches by adding a persistence/cache container that allows configuring cache sizes.
Some caches are already configurable, some are not. For instance, persistence/ps-cache-size can be used to set the psCache size. Other persistent caches are not configurable in size Session.cache, FastFindCache.cache and other. Double check if configuring the size makes sense. Default to their current value in the code when making them configurable. Use TempTableDataSourceProvider static block as example.

Implement support for a single container which allows the configuration of caches. Include there the existing configurable caches. Extract the caches which are not configurable and make them configurable.

#2 - 05/29/2023 03:25 AM - Alexandru Lungu

- Related to Bug #7330: Increase psCache size added

#3 - 06/07/2023 03:13 AM - Alexandru Lungu

- Assignee set to Dănuț Filimon

Danut, please take time to navigate through the persistence / orm package of FWD and search for any cache which can have its size configured externally. Report back with a list of whatever seems reasonable to configuration. Session.cache, FastFindCache and psCache are a place to start (the last one is already configurable).

#4 - 06/07/2023 09:07 AM - Dănuț Filimon

- File 7388-20230607.patch added

I found the following to be configurable:

- BufferManager.convertedNames, size is 10000;
- DynamicTablesHelper.cache, size is 8192;
- FastFindCache.l2Cache, size is 10, created when FastFindCache.put() is called;

- FastFindCache.l3Cache, size is 100, same situation as FastFindCache.l2Cache;
- FQLHelperCache.cache, size is 8192;
- FQLPreprocessor.cache, size is 2048;
- Persistence.Context.staticQueryCache, size is 1000;
- Persister.updateCache, size is 4096;
- Session.cache, size is 1024, created when Session.createCache() is called;

I attached a patch that takes the cache sizes from the configuration. Please review.

Eric, should this configuration be documented in the Wiki?

#5 - 06/07/2023 03:35 PM - Eric Faulhaber

Dănuț Filimon wrote:

> I found the following to be configurable:
>
> - BufferManager.convertedNames, size is 10000;
> - DynamicTablesHelper.cache, size is 8192;
> - FastFindCache.l2Cache, size is 10, created when FastFindCache.put() is called;
> - FastFindCache.l3Cache, size is 100, same situation as FastFindCache.l2Cache;
> - FQLHelperCache.cache, size is 8192;
> - FQLPreprocessor.cache, size is 2048;
> - Persistence.Context.staticQueryCache, size is 1000;
> - Persister.updateCache, size is 4096;
> - Session.cache, size is 1024, created when Session.createCache() is called;
>
> I attached a patch that takes the cache sizes from the configuration. Please review.

Code review 7388-20230607.patch:

This is a useful set of changes, thank you.

Can we please simplify the directory-reading code just a bit? For example, are the intermediate variables (e.g., convertedNamesCacheSize in BufferManager) actually needed? I would rewrite the static initializer as:

```
   static
   {
      DirectoryService ds = DirectoryService.getInstance();
      if (!ds.bind())
      {
         throw new RuntimeException("Directory bind failed");
      }

      try
      {
         String path = "persistence/cache-size/buffer-manager";
         CONVERTED_NAMES_CACHE_SIZE = Utils.getDirectoryNodeInt(ds, path, 10000, false);
      }
      finally
      {
         ds.unbind();
      }
   }
```

It might be argued that by using the extra local variable and setting the CONVERTED_NAMES_CACHE_SIZE in the finally block (as in the patch), CONVERTED_NAMES_CACHE_SIZE is set to a sensible default no matter how we exit the try block. However, if we exit the try in an exceptional way, the class isn't loading anyway and we have bigger problems, so I'd rather keep the code more concise. If I've missed some benefit of the original technique, please let me know.

Where possible, please make the cache instance variables final.

In some cases, the names of the configuration nodes in the directory are a little vague; they do not really describe the purpose of the cache. While we don't want these to be overly long, a directory path of persistence/cache-size/persistence doesn't tell you that it's the configuration setting for the size of the static query cache.

In FQLHelperCache, please update the comment for the cache from "LFU cache of FQL helper objects, which also ages its entries" to "Expiry cache of FQL helper objects". I missed changing this comment when I changed that cache from an LFU to LRU implementation.

In FastFindCache, we are defaulting the size of the L3 cache to 10, but I think the intent was 100.

> Eric, should this configuration be documented in the Wiki?

Yes, here: Database Configuration.

#6 - 06/08/2023 03:44 AM - Dănuț Filimon

I did a few changes as you suggested:

- Changed the name of a few cache nodes;
- Simplified the directory-reading code for: BufferManager, DynamicTablesHelper, FastFindCache, FQLHelperCache, Persistence. I did not make any changes to FQLPreprocessor and Persister since the cache is initialized in the static block, and Session because the configuration is read only in runtime mode;
- L3 cache value was supposed to be 100;
- Made the caches in BufferManager and Persister final;

Alexandru, where should these changes go?

#7 - 06/08/2023 03:47 AM - Alexandru Lungu

- Status changed from New to WIP

- % Done changed from 0 to 80

I created 7388a. Please commit your changes there. If everything is alright, I will do the merge.

#8 - 06/08/2023 04:02 AM - Constantin Asofiei

Please see #7398 for issues we have with the admin console, related to these static constructors which read from directory.xml.

#9 - 06/08/2023 04:26 AM - Alexandru Lungu

Constantin Asofiei wrote:

> Please see #7398 for issues we have with the admin console, related to these static constructors which read from directory.xml.

Do you mean we should bootstrap this approach? Therefore, should we set this cache sizes at run-time so that we avoid binding the configuration file at class loading - allowing us to do work **before** actually reading the configuration?
I am not quite sure what is the issue right now.

#10 - 06/08/2023 04:55 AM - Constantin Asofiei

Yes, reading this at server startup will solve this.

The problem is that directory is being read while another code has a directory batch operation open. And it breaks this case.

#11 - 06/08/2023 05:26 AM - Dănuț Filimon

- Assignee changed from Dănuț Filimon to Eric Faulhaber

- vendor_id deleted (GCD)

I already committed 7388a.rev14618, but will make the changes to read the cache size values at runtime.

#12 - 06/08/2023 05:26 AM - Dănuț Filimon

- Assignee changed from Eric Faulhaber to Dănuț Filimon

- vendor_id set to GCD

#13 - 06/08/2023 06:57 AM - Alexandru Lungu

- Assignee changed from Dănuț Filimon to Eric Faulhaber

- vendor_id deleted (GCD)

Constantin Asofiei wrote:

> Yes, reading this at server startup will solve this.
>
> The problem is that directory is being read while another code has a directory batch operation open. And it breaks this case.

Makes sense. However, this may be quite odd as we will need a .initialize method which does this whole cache work:

- either we have Session.initialize, TempTableDataSourceProvider.initialize, etc. (i.e. one initialize per class using a cache). This may get out of hand as there are several places that intent to use the directory, so we may have lots and lots of .initialize which may be hard to manage.
- for this case precisely, we can have a (Persistence)?CacheManager which does this initialization for all places we have caches. Basically, on DatabaseManager.initialize, CacheManager.initialize will be called and all cache sized through-out FWD will be set. However, this means:
    - There is a central cache manager which will be aware of **all** caches and knows to search the right directory node and set the right cache size **for each** cache.
    - There should be a default value for the cache sizes and they won't be final anymore.

From my POV, most interesting option is: we can have a factory for these caches inside CacheManager, so (for example Session) can call CacheManager.createCache(Session.class) to retrieve a correctly sized and configured cache. We can even convey that cache size 0 actually means no cache. At this point, we can use the class name in directory.xml (just like we do with security/resource-instance) to refer the right cache. This still has the flaw: we can't easily configure multiple caches defined in a single class - we still need a string discriminator here.

This looks pretty neat; Constantin/Eric, what is your input on this?

#14 - 06/08/2023 06:58 AM - Alexandru Lungu

- Assignee changed from Eric Faulhaber to Dănuț Filimon

- vendor_id set to GCD

This is strange ...

#15 - 06/08/2023 10:19 AM - Eric Faulhaber

Alexandru Lungu wrote:

> This is strange ...

A hard reload of the browser page before adding a task history entry avoids this. Not ideal, but it's a workaround.

#16 - 06/08/2023 11:34 AM - Eric Faulhaber

Alexandru Lungu wrote:

> Constantin Asofiei wrote:
>
> > Yes, reading this at server startup will solve this.
> >
> > The problem is that directory is being read while another code has a directory batch operation open. And it breaks this case.
>
> Makes sense. However, this may be quite odd as we will need a .initialize method which does this whole cache work:
>
> - either we have Session.initialize, TempTableDataSourceProvider.initialize, etc. (i.e. one initialize per class using a cache). This may get out of hand as there are several places that intent to use the directory, so we may have lots and lots of .initialize which may be hard to manage.
> - for this case precisely, we can have a (Persistence)?CacheManager which does this initialization for all places we have caches. Basically, on DatabaseManager.initialize, CacheManager.initialize will be called and all cache sized through-out FWD will be set. However, this means:
>   - There is a central cache manager which will be aware of **all** caches and knows to search the right directory node and set the right cache size **for each** cache.
>   - There should be a default value for the cache sizes and they won't be final anymore.
>
> **From my POV, most interesting option is**: we can have a factory for these caches inside CacheManager, so (for example Session) can call CacheManager.createCache(Session.class) to retrieve a correctly sized and configured cache. We can even convey that cache size 0 actually means no cache. At this point, we can use the class name in directory.xml (just like we do with security/resource-instance) to refer the right cache. This still has the flaw: we can't easily configure multiple caches defined in a single class - we still need a string discriminator here.
>
> This looks pretty neat; Constantin/Eric, what is your input on this?

I don't love the idea of divorcing the creation of the caches from the class that uses them, though centralization of the directory reading sounds good to me. Continuing with the Session example, here is the method which creates the cache:

```
   private ExpiryCache<RecordIdentifier<String>, BaseRecord> createCache()
   {
      ExpiryCache<RecordIdentifier<String>, BaseRecord> c =
         new LRUCache<>(database.toString() + " database session cache",
                        SESSION_CACHE_SIZE,
                        (dmo) -> !dmo.isInUse(),
                        CapacityPolicy.LENIENT);
      c.addCacheExpiryListener(this);

      return c;
   }
```

All of the information being passed to the cache c'tor is useful to have within the Session class, close to where it is being used. Routing this through a factory seems messy. I could see having all the directory inspection code in one class (PersistenceCacheConfig or something similar), and having the cache users ask that class for the configured size. That class would be rather dumb (primarily just for centralizing the directory reading). Assuming the directory access happens during server bootstrap, before any of the user classes can interrogate it, we would not need any synchronization.

If I've misunderstood your proposal, please elaborate.

#17 - 06/09/2023 02:55 AM - Alexandru Lungu

Eric Faulhaber wrote:

> Routing this through a factory seems messy

I think it is easier to convey that a CacheManager should always produce the caches, rather than always remember to use PersistenceCacheConfig for size instead of hard coding it. From my POV, a unified cache factory doesn't sound that bad to me - such technique was used widely before with loggers Logger log = LogHelper.getLogger([...]) (and currently with CentralLogger). Moreover, we can have **all** caches configurable through diretory.xml out-of-the-box if a CacheManager is used. We can even enforce caches with SoftReference (to be resilient with leaks and OOM) or proxy them to do logging/profiling.

Making the LRUCache c'tor package-private to enforce the CacheManager policy may seem messy indeed.

I am thinking bigger than only setting the size :)

> If I've misunderstood your proposal, please elaborate.

I was mostly thinking of keeping the same API: instead of new LRUCache, use CacheManager.create.

```
ExpiryCache<RecordIdentifier<String>, BaseRecord> c =
         CacheManager.create(Session.class, // configuration identifier
                             database.toString(), // discriminator (not mandatory)
                             1024, // default cache size (overridden by whatever is configured)
                             (dmo) -> !dmo.isInUse(), // quite custom (can't be overridden)
                             CapacityPolicy.LENIENT) // default policy (overridden by whatever is configured)
```

I was thinking of something like the above. My point is that any setting can be configured, not only the size:

- The name will always be something like <identifier>(-<discriminator>)?
- The cache size is configured by Session, but can be overridden by CacheManager
- The policy is configured by Session, but can be overridden by CacheManager
- Maybe we can even customize the type of cache (LRUCache vs LFUAgingCache)

In essence, CacheManager is a factory that allows the override of this settings if any configuration is found. I kept CacheManager name inline with the other managers (e.g. DatabaseManager), but we can change it to CacheFactory. Also, note that CacheManager is only a server runtime thing, so the conversion won't make use of such settings.

Your suggestion leads to:

```
ExpiryCache<RecordIdentifier<String>, BaseRecord> c =
         new LRUCache<>(database.toString() + " database session cache",
                        PersistenceCacheConfig.getOrDefault(Session.class, 1024), // this is the only change
                        (dmo) -> !dmo.isInUse(),
                        CapacityPolicy.LENIENT);
```

I guess this is the cleanest approach to handle **only** configurable cache sizes for a limited number of caches.

> Assuming the directory access happens during server bootstrap, before any of the user classes can interrogate it, we would not need any synchronization.

The synchronization is not needed, no matter the approach. The initialization will be done at server bootstrap.

#18 - 06/09/2023 04:35 AM - Alexandru Lungu

- Related to Bug #7425: Avoid directory usage in static blocks added

#19 - 06/09/2023 10:15 AM - Eric Faulhaber

Alexandru Lungu wrote:

> I am thinking bigger than only setting the size :)

Thinking beyond the current problem is good :)

> I was mostly thinking of keeping the same API: instead of new LRUCache, use CacheManager.create.
>
> [...]

This looks ok to me.

> I was thinking of something like the above. My point is that any setting can be configured, not only the size:

I don't think **every** setting should be configurable. Generally speaking, the developer will have the best idea about which is the most appropriate capacity policy (especially where that comes into play with an expiration test), because it affects functionality. An administrator will have better knowledge about the most appropriate size and name, assuming they understand the purpose of the cache.

> - The name will always be something like <identifier>(-<discriminator>)?

OK.

> - The cache size is configured by Session, but can be overridden by CacheManager

OK.

> - The policy is configured by Session, but can be overridden by CacheManager

In most cases, I think the policy needs to be chosen by the developer and should not be overridden.

Sticking with the Session example... the cache is not only about performance. If the policy is not lenient, the functionality can break. We cannot allow DMOs which are currently "in use" to be expired, even if that means the cache must grow temporarily. Otherwise, we may get duplicate DMOs in memory for the same record, which has negative repercussions in the persistence framework. A strict policy does not work here. We don't want to give an administrator an option which, if chosen, is known to break the system.

> - Maybe we can even customize the type of cache (LRUCache vs LFUAgingCache)

Theoretically, yes. In pragmatic terms, however, even though I like the idea of LFUAgingCache better than LRUCache for a lot of use cases, the implementation of LFUAgingCache is much too slow for practical use.

> In essence, CacheManager is a factory that allows the override of this settings if any configuration is found. I kept CacheManager name inline with the other managers (e.g. DatabaseManager), but we can change it to CacheFactory. Also, note that CacheManager is only a server runtime thing, so the conversion won't make use of such settings.

I am fine with either name. If this is meant to be a general-purpose facility (i.e., not just persistence), we need to rethink the organization of the container(s) in the directory.

You have persuaded me that a cache manager/factory implementation can make sense; hopefully my restrictions above do not make the implementation or use too complicated.

#20 - 06/09/2023 10:48 AM - Alexandru Lungu

Eric Faulhaber wrote:

> I don't think **every** setting should be configurable. Generally speaking, the developer will have the best idea about which is the most appropriate capacity policy (especially where that comes into play with an expiration test), because it affects functionality. An administrator will have better knowledge about the most appropriate size and name, assuming they understand the purpose of the cache.
>
> Sticking with the Session example... the cache is not only about performance. If the policy is not lenient, the functionality can break. We cannot allow DMOs which are currently "in use" to be expired, even if that means the cache must grow temporarily. Otherwise, we may get duplicate DMOs in memory for the same record, which has negative repercussions in the persistence framework. A strict policy does not work here. We don't want to give an administrator an option which, if chosen, is known to break the system.

You are right. I messed up the meaning of LENIENT here. It thought that the lenient expiry = don't clear the cache eagerly, but expire record in batches. Now I see that this is in fact about cleanLimit, **which by the way** is not configurable yet :) However, this may be too granular.

> > - Maybe we can even customize the type of cache (LRUCache vs LFUAgingCache)
>
> Theoretically, yes. In pragmatic terms, however, even though I like the idea of LFUAgingCache better than LRUCache for a lot of use cases, the implementation of LFUAgingCache is much too slow for practical use.

I found quite interesting cache alternatives at some point (if I recall correctly, one was named "caffeine"). Of course, such solutions should closely investigated, tested and wrapped to be used. However, I don't deny that such integration may ever happen.

> > In essence, CacheManager is a factory that allows the override of this settings if any configuration is found. I kept CacheManager name inline with the other managers (e.g. DatabaseManager), but we can change it to CacheFactory. Also, note that CacheManager is only a server runtime thing, so the conversion won't make use of such settings.
>
> I am fine with either name. If this is meant to be a general-purpose facility (i.e., not just persistence), we need to rethink the organization of the container(s) in the directory.
>
> You have persuaded me that a cache manager/factory implementation can make sense; hopefully my restrictions above do not make the implementation or use too complicated.

I don't think this is that hard to achieve, right Danut? Of course, I guess we should go with a "fine-tuning" kind of container to manage these - or simply "caches".

Eric, what is your input on integrating SoftReference at this point? Danut, can you provide a feedback how easily is to integrate such feature considering the current status.

#21 - 06/09/2023 11:07 AM - Eric Faulhaber

There may be some specific use cases where they make sense, but in general, I am not a huge fan of soft references, for several reasons:

- I don't like having the benefit of a cache disappear, out of my control.
- In a server that uses as much memory as we do, reclaiming a soft reference may buy a small amount of time. But if the JVM is at the point where it is clearing soft references to survive, the system is thrashing for memory already, and you don't have that much time left before OOM anyway. There is either a leak to fix or the system just needs more heap at a steady state.

That being said, my experience with using SoftReference is from several JVM generations ago, so maybe something has changed to invalidate my conclusions. If you have some thoughts that I might not be considering, please let me know.

#22 - 06/12/2023 03:13 AM - Alexandru Lungu

Eric Faulhaber wrote:

> That being said, my experience with using SoftReference is from several JVM generations ago, so maybe something has changed to invalidate my conclusions. If you have some thoughts that I might not be considering, please let me know.

I see your point. I should also mention that SoftReference introduces a bit of code cognitive complexity as each time the cache is used, .get should be used and checked for null (+ performance complexity?).

> I don't like having the benefit of a cache disappear, out of my control.

It makes sense for caches which are heavily coupled to our code, like Session.cache. However, we can survive without prepared statement, FQLHelper or fast-find caches at the cost of extra performance. My point is that most caches are "optional", in the sense that FWD can survive without them. So thrashing between exists / doesn't exist is not something to worry about - maybe only for dev. time where we are debugging. Separating concerns (caches - business logic) should help us on this point.

> You don't have that much time left before OOM anyway.

There are two cases I am thinking of here:

- The sever is improperly configured in terms of memory in regard to the number of sessions expected. For example, FWD is started so that only 4 sessions can be supported in the memory limit. A 5-th / a more demanding session will cause OOM. In this case, we can save memory by dropping the caches, so we avoid a fatal crash. I expect the client-level caches to be demanding, although I don't have a statistic here.
- There is a leak. From previous experience, the leaks were caused by the caches (or cache alike structures). Therefore, dropping them may solve the leak. This is not optimal as we are basically hiding a flaw in FWD using SoftReference, but it can save a production environment from fatally crashing. However, we can detect such issues in a testing environment later on.

I am not a first class fan of SoftReference either, but this was stuck into my head since #7241.

#23 - 06/13/2023 06:31 AM - Dănuț Filimon

Committed 7388a.rev14619. Added CacheManager that reads the configuration cache sizes of classes and can be used to create caches.

The initial implementation can be summarized into a single method that reads the configuration of each cache from a file, the rest of the implementation depended on the classes that were subjected to this change, and it took me some time to understand and compile each one of them.

Here is a summary of the changes:

- Remove the static directory usage, now the cache sizes are obtained when the CacheManager.initialize() is called and the directory service can be accessed.
- Static caches are instantiated using the initializeCache method of their specific class.
- Non static caches are instantiated in the class constructor and there are no longer final caches. (Caches that are initialized in the constructor can be final, I will revert this change soon).
- Caches can be created using CacheManager.createLRUCache and CacheManager.createMapCache.
- The directory configuration contains a container for each cache, and it will have the class name, the size and an optional discriminator (it can be null for classes that have a single cache).

Please review and advise.

#24 - 07/05/2023 04:38 AM - Constantin Asofiei

- Related to Feature #6815: configure all cache sizes in the directory, and create documentation for them added

#25 - 07/06/2023 09:22 AM - Alexandru Lungu

- % Done changed from 80 to 100

- Status changed from WIP to Review

Merged 7388a in trunk as rev. 14641 and archived.

#26 - 07/18/2023 02:18 PM - Greg Shah

- Status changed from Review to Test

Files

7388-20230607.patch	19.8 KB	06/07/2023	Dănuț Filimon